Lecture 7: Overview

- Selection
- Greedy Algorithms (Chapter 5)
- Dynamic Programming
- Check email for new homework assignment and programming assignment
Selecting the ith Element

• We want to select the ith smallest element
• Can use quicksort type algorithm (with randomized node selection)
• Basic idea:
  1. Select random pivot point
  2. Split nodes around pivot point
  3. Run procedure on set of nodes that contains ith smallest element
Complexity Analysis

- **Best Case**: $O(n)$ - pivot point is the $i$th element
- **Even partition**: 
  \[ T(n) = O(n) + O(n/2) + O(n/4) + \ldots + O(2) + O(1) = O(2n) = O(n) \]
- **Worst case**: 
  \[ T(n) = O(n) + O(n-1) + \ldots + O(2) + O(1) = O(n^2) \]
Complexity Analysis

- **Average time:**
  
  1/n probability of each position i as pivot.

\[ T(n) = \sum_{i=1}^{n-1} \frac{1}{n} T(\max(i, n-i)) + O(n) \]

\[ = n \sum_{i=[n/2]}^{n-1} \frac{2}{n} T(i) + O(n) \]

Guess \( T(n) \leq cn \)

\[ T(n) \leq \frac{2}{n} \sum_{i=[n/2]}^{n-1} ci + an = \frac{2c}{n} \left( \sum_{k=1}^{n} k - \sum_{k=1}^{[n/2]-1} k \right) + an \]

\[ = \frac{2c}{n} \left( \frac{(n-1)n}{2} - \frac{([n/2]-1)[n/2]}{2} \right) + an \leq \frac{2c}{n} \left( \frac{(n-1)n}{2} - \frac{(n/2 - 2)(n/2 - 1)}{2} \right) \]

\[ = c \left( \frac{3n}{4} + \frac{1}{2} - \frac{2}{n} \right) + an \leq cn - \left( \frac{cn}{4} - \frac{c}{2} - an \right) \]
Complexity Analysis

• Need to show that for large n, last expression is at most cn. To do this, \( cn/4-c/2-an \geq 0 \).
• Choose \( c>4a \)
• Gives \( n \geq 2c/(c-4a) \)
Select w/ worst-case linear time

- Idea: split input into sets of 5
- Compute median of each of these sets
- Use select to compute the median of these medians
- Pivot around this point
Select w/ linear worst-case

- Pivot point is guaranteed to be good:
- Note that half of medians are less than or equal to the chosen median of medians and half are greater or equal
- Each of these sets contributes either 3 elements that are less than or equal to the median of medians or 3 elements that are greater than or equal to the median of medians
- Book contains proof that this algorithm is $O(n)$
Greedy Algorithms

• Solve a global optimization problem by making localized (or greedy decisions)

• Example: Making change (choose the largest coin)
  • Works for US currency
  • Doesn’t work for all possible coins: Consider using 0.30, 0.20, 0.05, 0.01 to make change for 0.40
Knapsack Problems

• Fractional Knapsack Problem
  • $i$th item is worth $v_i$ dollars and weighs $w_i$
  • If the knapsack can hold a maximum weight $W$, how do we choose items
  • Can load fractions of an item
  • Strategy - fill with most valuable items (per unit weight), then next...
Activity Selection Problem

- Set $S$ of $n$ activities
- Each activity has
  - $s_i = \text{start time of activity } i$
  - $f_i = \text{finish time of activity } i$
  - $0 \leq s_i < f_i < \infty$
- Activities $i$ and $j$ are compatible
  - if $f_i \leq s_j$ or $f_j \leq s_i$ (no overlap)
- Problem: select the maximum size subset of mutually compatible activities
Structure of the Optimal Solution

- $S_{ij} = \{ a_k \in S : f_i \leq s_k < f_k \leq s_j \}$
- Let $A =$ optimal set for $S$
- $A_{i,j} =$ optimal for $S_{i,j}$
- if $a_k \in A_{i,j}$ then
  - $A_{i,k}$ must be optimal solution to $S_{i,k}$
  - Proof: assume there exist $A'_{i,k}$ with more activities than our $A_{i,k}$ (that is $|A'_{i,k}| > |A_{i,k}|$
  => we can construct a longer $A'_{i,j}$ by using $A'_{i,k}$ prefix. Contradiction.
Solution Algorithm

- Let $c[i,j] = \max$ # of compatible activities in $S_{i,j}$
- Want to maximize $C[0,n+1]$
- $c[i,j] = 0$ if $S_{i,j} = \emptyset$
  
  
  $= \max\{c[i,k]+c[k,j]+1\}$ for $i<k<j$ is $S_{i,j} \neq \emptyset$

- Can do better with greedy choice

- Solution:
  Pick next compatible task with earliest finishing time
Matrix-Chain Multiple

- Matrix multiply is associative
  - $(AB)C = A(BC)$
  - Computation cost can be different (dimensions differ)
- Problem: How to optimize placement of parentheses to minimize cost
Consider the product: $A_1 A_2 \ldots A_n$

Let $A_{ij}$ denote the product $A_i A_{i+1} \ldots A_{j-1} A_j$

We want to choose $k$ such that computing $A_{ik} A_{k+1} j$ is the minimal way to compute $A_{ij}$

Let $M(i,j)$ denote the cost of computing $A_{ij}$

$M(i,j) = 0$ if $i = j$ or $\min(M(i,k)+M(k+1,j)+p_{i-1}p_k p_j)$ for $i \leq k < j$ if $i < j$
Recursive Procedure

\[ M(i,j) \]
if \( i = j \)
then return 0
else
\[ \text{min} = M(i,i) + M(i+1,j) + p_{i-1}p_ip_j \]
for \( k \leftarrow i+1 \) to \( j-1 \)
\[ \begin{align*} 
\text{if } \text{min} &> M(i,k) + M(k+1,j) + p_{i-1}p_kp_j \\
\text{min} &= M(i,k) + M(k+1,j) + p_{i-1}p_kp_j 
\end{align*} \]
return min
Problem

- This recursive implementation is expensive
- We need to cache $M(i,j)$'s instead of re-computing them
- Can compute from the bottom up
0-1 Knapsack Problem

- Whole items only
- ith item is worth $v_i$ dollars and weighs $w_i$
- If the knapsack can hold a maximum weight $W$, how do we choose items
- Can’t load fractions of an item
- Greedy algorithm doesn’t work
- Trying all combinations too computationally expensive
Optimal Structure

- If we remove \( w_j \) the remain load must be the most value load weighing \( W-w_j \) that can be taken from the items excluding \( j \)
- Greedy strategy works for fractional problem but not 0-1 problem
Optimal Structure

\[ B[k,w] = B[k-1, w] \text{ if } w_k > w \]
\[ \max\{B[k-1, w], B[k-1, w-w_k] + b_k\} \]
otherwise
**Algorithm**

Input: Set S of n items such that item i has positive benefit $b_i$ and weight $w_i$. Total weight $W$.
Output: For $w=0,\ldots,W$, maximum benefit $B[w]$ of a subset of S with total weight at most $w$

for $w<-0$ to $W$ do
  $B[w]<-0$
for $k<-1$ to $n$ do
  for $w<-W$ downto $w_k$ do
    if $B[w-w_k]+b_k > B[w]$ then
      $B[w]<-B[w-w_k]+b_k$